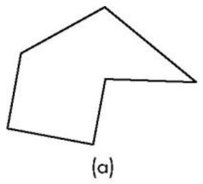
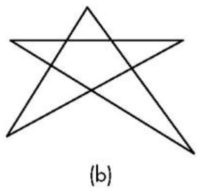
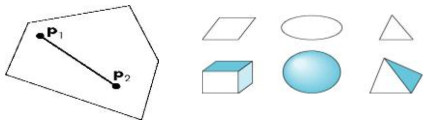
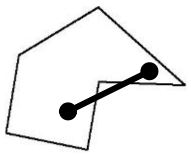
Sheet 3 solution

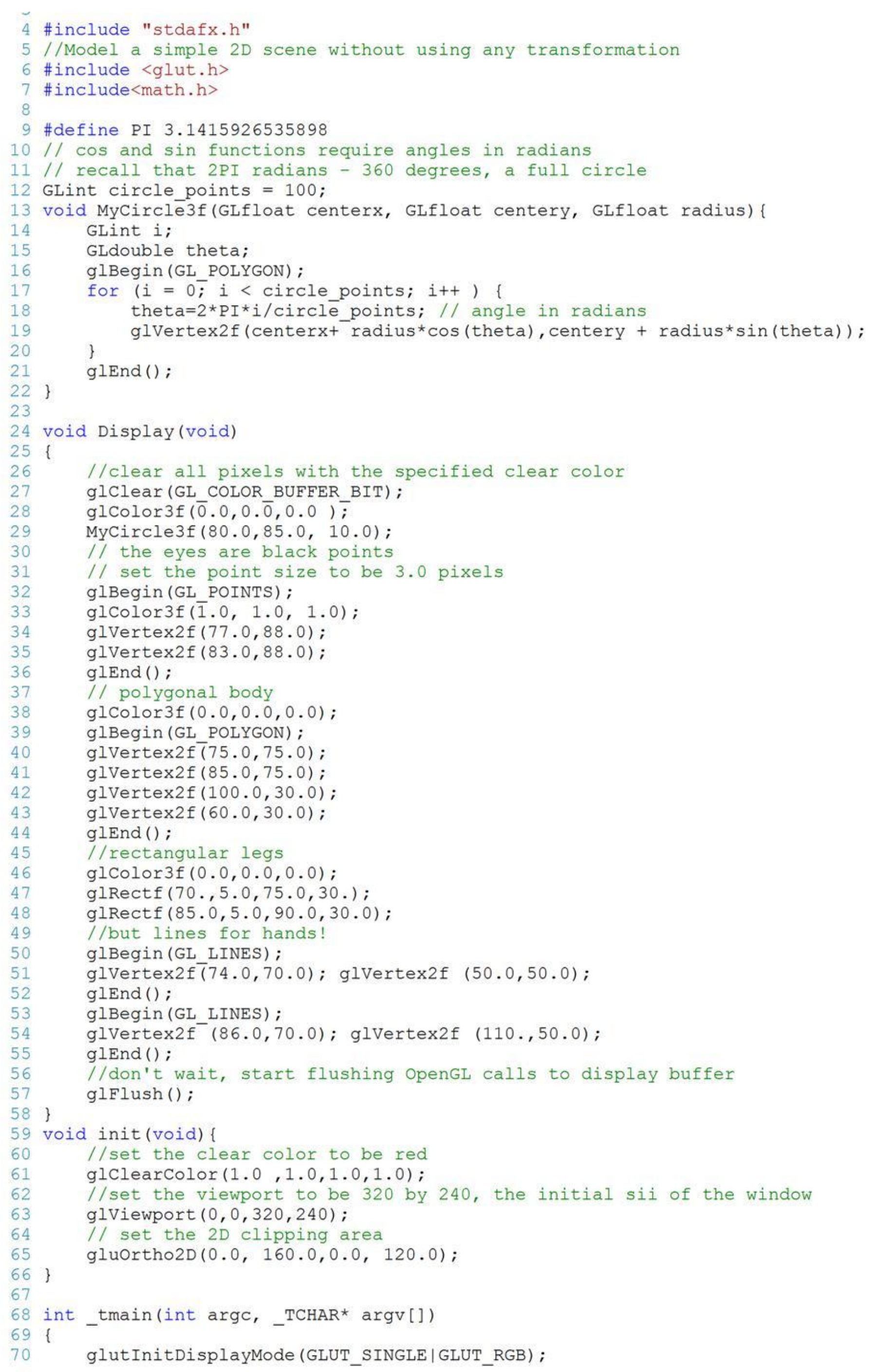
1. **Primitive functions:** are functions that draw primitive objects like points, lines, triangles and polygons
   Attribute functions: are functions that specify how the primitives are drawn. For examples, the functions that specify the current drawing color and the current clear color are examples of the attribute functions
   Viewing functions: are functions that specify how the objects are viewed. They specify the location of the viewer in the world, the view direction, and the type of projection.
   Transformation functions: are functions that allow us performing transformation on object before drawing them. Examples of these transformations are rotation, scaling, and translation.
   Input functions: are functions that allow the user to input some data to the graphics program while being executed. These functions are used to add interactivity to the program. Examples are functions that allow the user to specify a location using the mouse or input string/numeric data using the keyboard
   Control function: users. The control functions enable us to communicate with the window system, to initialize our programs, and to deal with any errors that take place during the execution of our programs.
   Query functions: are functions that provide information about the current setting of the graphics library.

2. Three properties will ensure that a polygon will be displayed correctly: It must be simple, convex, and flat.
   - Simple: as long as no two edges of a polygon cross each other, we have a simple polygon. Testing is very complex and is left to the application program not to the gl lib.
   - Convex: An object is convex if all points on the line segment between any two points inside the object, or on its boundary, are inside the object. Testing is very complex and is left to the application program not to the gl lib.
   - Flat: The entire object lies in the same plane

   The following figures explain the simplicity and the convexity of a polygon

  (a) simple, (b) non simple	(a)  (b)  Convex polygon and objects (a) and non-convex polygon (b)
Simplicity	Convexity

3. (a)

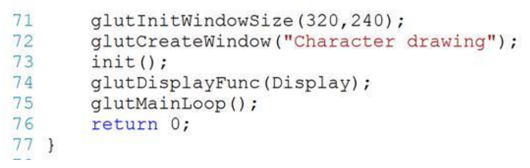


```
#include "stdafx.h"
//Model a simple 2D scene without using any transformation
#include <glut.h>
#include<math.h>

#define PI 3.1415926535898
// cos and sin functions require angles in radians
// recall that 2PI radians - 360 degrees, a full circle
GLint circle_points = 100;
void MyCircle3f(GLfloat centerx, GLfloat centery, GLfloat radius){
    GLint i;
    GLdouble theta;
    glBegin(GL_POLYGON);
    for (i = 0; i < circle_points; i++ ) {
        theta=2*PI*i/circle_points; // angle in radians
        glVertex2f(centerx+ radius*cos(theta),centery + radius*sin(theta));
    }
    glEnd();
}

void Display(void)
{
    //clear all pixels with the specified clear color
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0,0.0,0.0 );
    MyCircle3f(80.0,85.0, 10.0);
    // the eyes are black points
    // set the point size to be 3.0 pixels
    glBegin(GL_POINTS);
    glColor3f(1.0, 1.0, 1.0);
    glVertex2f(77.0,88.0);
    glVertex2f(83.0,88.0);
    glEnd();
    // polygonal body
    glColor3f(0.0,0.0,0.0);
    glBegin(GL_POLYGON);
    glVertex2f(75.0,75.0);
    glVertex2f(85.0,75.0);
    glVertex2f(100.0,30.0);
    glVertex2f(60.0,30.0);
    glEnd();
    //rectangular legs
    glColor3f(0.0,0.0,0.0);
    glRectf(70.,5.0,75.0,30.);
    glRectf(85.0,5.0,90.0,30.0);
    //but lines for hands!
    glBegin(GL_LINES);
    glVertex2f(74.0,70.0); glVertex2f (50.0,50.0);
    glEnd();
    glBegin(GL_LINES);
    glVertex2f (86.0,70.0); glVertex2f (110.,50.0);
    glEnd();
    //don't wait, start flushing OpenGL calls to display buffer
    glFlush();
}
void init(void){
    //set the clear color to be red
    glClearColor(1.0 ,1.0,1.0,1.0);
    //set the viewport to be 320 by 240, the initial sii of the window
    glViewport(0,0,320,240);
    // set the 2D clipping area
    gluOrtho2D(0.0, 160.0,0.0, 120.0);
}

int _tmain(int argc, _TCHAR* argv[])
{
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
```

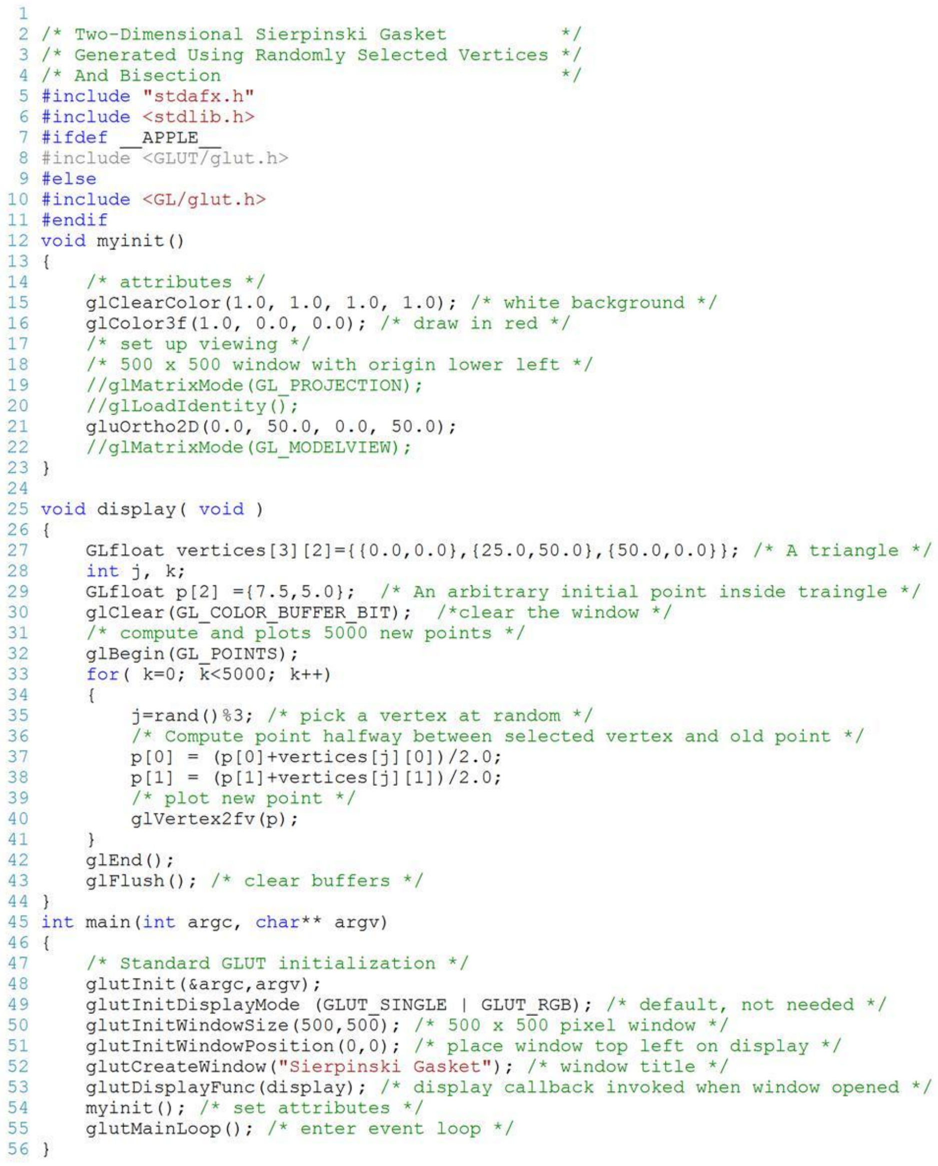


```
71     glutInitWindowSize(320,240);
72     glutCreateWindow("Character drawing");
73     init();
74     glutDisplayFunc(Display);
75     glutMainLoop();
76     return 0;
77 }
```

(b)

The same as the above program with the addition of the line:
”`glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);`”
After the line number: 38

4.



```
/* Two-Dimensional Sierpinski Gasket          */
/* Generated Using Randomly Selected Vertices */
/* And Bisection                              */
#include "stdafx.h"
#include <stdlib.h>
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
void myinit()
{
    /* attributes */
    glClearColor(1.0, 1.0, 1.0, 1.0); /* white background */
    glColor3f(1.0, 0.0, 0.0); /* draw in red */
    /* set up viewing */
    /* 500 x 500 window with origin lower left */
    //glMatrixMode(GL_PROJECTION);
    //glLoadIdentity();
    gluOrtho2D(0.0, 50.0, 0.0, 50.0);
    //glMatrixMode(GL_MODELVIEW);
}

void display( void )
{
    GLfloat vertices[3][2]={{0.0,0.0},{25.0,50.0},{50.0,0.0}}; /* A triangle */
    int j, k;
    GLfloat p[2] ={7.5,5.0};  /* An arbitrary initial point inside traingle */
    glClear(GL_COLOR_BUFFER_BIT);  /*clear the window */
    /* compute and plots 5000 new points */
    glBegin(GL_POINTS);
    for( k=0; k<5000; k++)
    {
        j=rand()%3; /* pick a vertex at random */
        /* Compute point halfway between selected vertex and old point */
        p[0] = (p[0]+vertices[j][0])/2.0;
        p[1] = (p[1]+vertices[j][1])/2.0;
        /* plot new point */
        glVertex2fv(p);
    }
    glEnd();
    glFlush(); /* clear buffers */
}
int main(int argc, char** argv)
{
    /* Standard GLUT initialization */
    glutInit(&argc,argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); /* default, not needed */
    glutInitWindowSize(500,500); /* 500 x 500 pixel window */
    glutInitWindowPosition(0,0); /* place window top left on display */
    glutCreateWindow("Sierpinski Gasket"); /* window title */
    glutDisplayFunc(display); /* display callback invoked when window opened */
    myinit(); /* set attributes */
    glutMainLoop(); /* enter event loop */
}
```